**Operating System HW5**

**학번:** 2016113832

**이름:** 윤동준

**담당교수:** 강순주

1. **개요**

첫번째 과제1번은 user프로그램에서 키보드 입력 A또는 B를 입력받아서 led를 제어하는 프로그램을 만들었고 과제2번은 led\_module를 수정하여 회로 내에서 switch를 누르면 그 값을 device driver에 저장하여 LED를 키게 하였다.

1. **프로그램 구조 설명**

**2.1 함수에 대한 설명**

**-int start\_module(void):** insmode led\_module, 모듈을 시작할 때 사용하는 함수

**-void end\_module(void) :** rmmode led\_module, 모듈을 종료할 때 사용하는 함수

**-static int gpio\_open(struct inode \*inode, struct file \*file):** 응용 프로그램에서(사용자 프로그램) 장치 파일을 system call을 호출해서 열 때, 커널이 호출하는 함수이다.

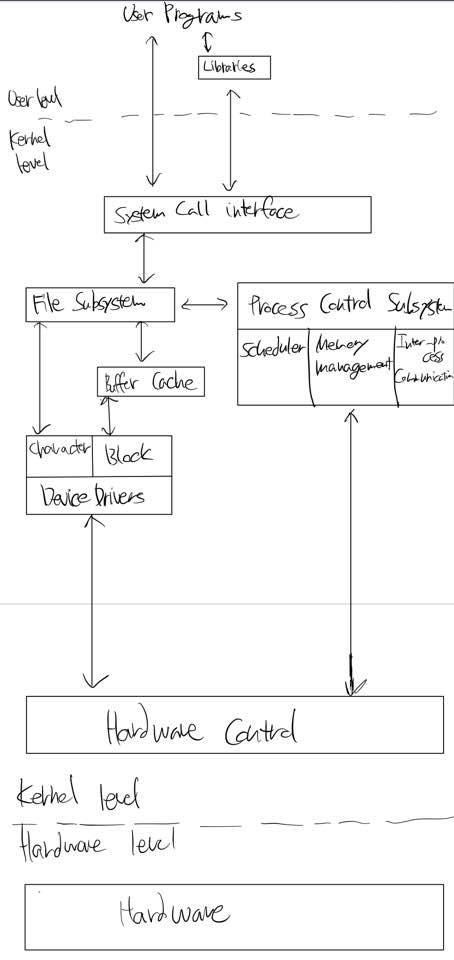
-**static int gpio\_release(struct inode\* inode, struct file\* file):** 응용 프로그램에서(사용자 프로그램) 장치 파일을 system call을 호출해서 닫을 때, 사용한다.

**-static ssize\_t gpio\_read(struct file\* file, char\* buf, size\_t len, loff\_t\* off):** 응용 프로그램에서(사용자 프로그램) 장치 파일을 system call을 호출해서 읽을 때, 사용한다.

-**static ssize\_t gpio\_write(struct file\* file, const char\* buf, size\_t len, loff\_t\* off):** (과제1번)응용 프로그램에서(사용자 프로그램) 장치 파일을 system call을 호출해서 쓸 때, 사용한다. (과제2번) gpio\_get\_value를 사용하여 switch를 눌렀을 때 값을 받아와서, 1이면 led를 On시키고, 0이면 led를 Off시킨다.

-**void main(int argc, char\*\* argv):** (과제1번) 메인문에서 if문을 사용하여 A를 입력받으면 usleep를 통해 0.5초 딜레이를 두고 write를 사용하여 led가 켜지고 꺼짐을 3번 반복한다. (과제2번) 스위치 값이 들어가기 전까지 write(fd, NULL, NULL)을 지속적으로 넣어줌으로써 아무 행동없이 대기하게 하였다.

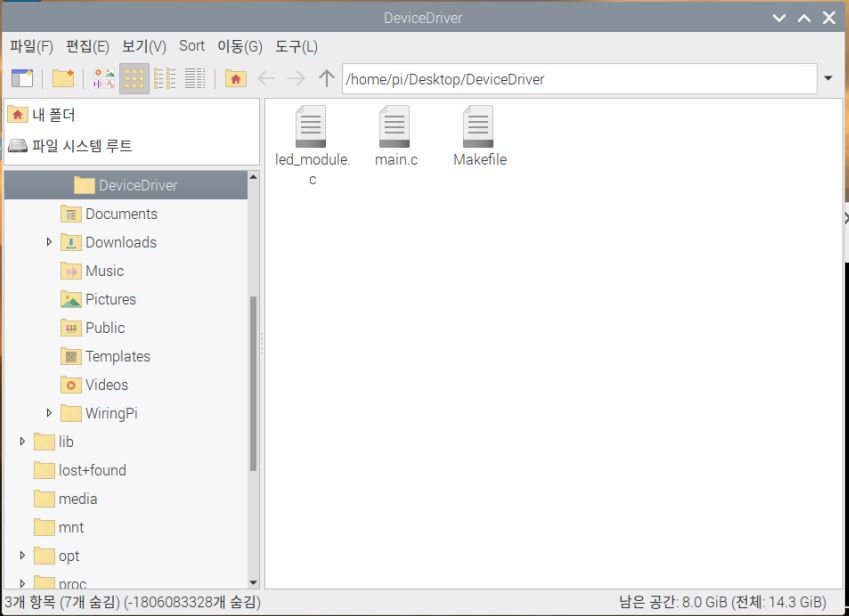
**다이어그램**



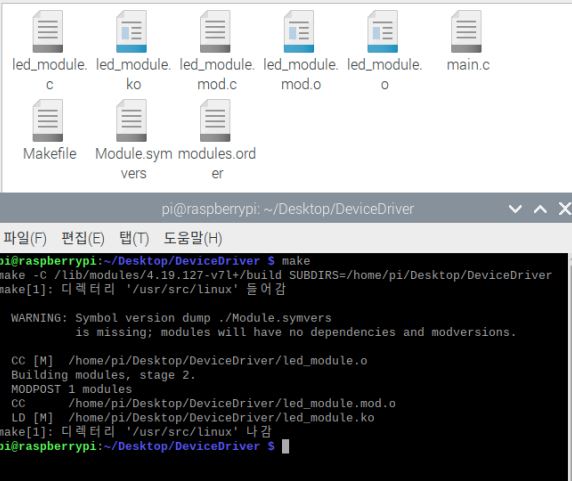
|  |  |
| --- | --- |
| **커널함수** | **유저함수** |
| start\_module | insmod |
| gpio\_open | fopen() |
| gpio\_read/gpio\_write |  |
| gpio\_close | fclose() |
| end\_module | rmmod |

1. **실행결과 (동영상 확인)**

**과제1번**



Make 명령어를 수행하기 전 모습이다.



Make 명령어 수행 후 파일들이 여러 개 만들어졌다.

주어진 코드에서 #define GPIO\_MAJOR 245로 되어 있는 것을 243으로 수정하고, GPIO 26으로 설정하였다. 유저모드에서 A와 B값을 받아서 출력을 내야 해서, main.c에 scanf문을 사용하여 scan된 값이 A이면 0.5초 주기로 3번 깜빡이는 것을 확인하였고, B값이면 그 후 한 번 더 입력을 받아서 1을 입력하면 led가 켜지고, 0을 입력하면 led가 꺼지는 것을 확인하였다.

**과제2**

위와 같이 make 명령어 수행 후 스위치버튼을 눌러서 스위치의 값을 gpio\_get\_value를 통하여 받은 후, 그 값이 1이면 led에 불을 들어오는 것을 확인하였고, 그 값이 0이면 led에 불이 들어오지 않는 것을 확인하였다. (동영상)

For반복문을 사용하였을 때, 3번 깜빡임을 위해 3번으로 했지만, 한 번밖에 깜빡이지 않았다. 확인해보니 for (int j=0; j<3; j++) {

write(fd, "1", 1);

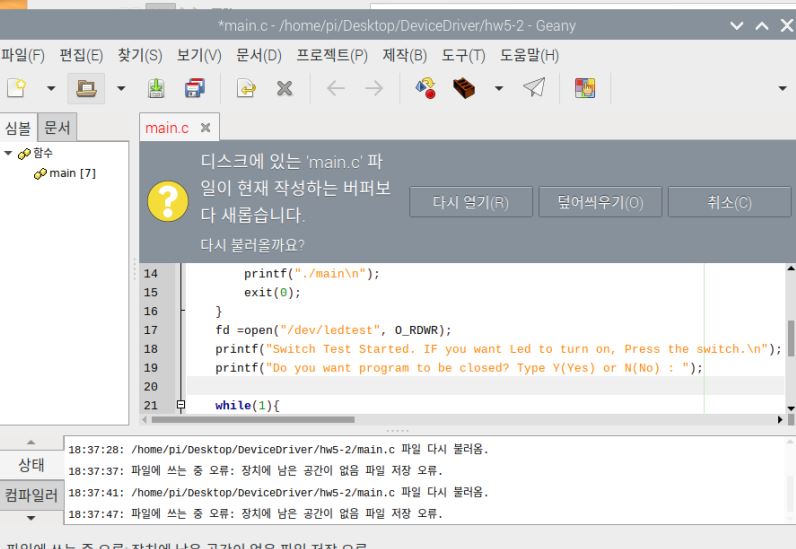
usleep(500000);

write(fd, "0", 1);

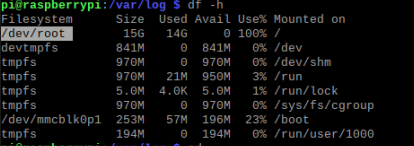
usleep(500000);

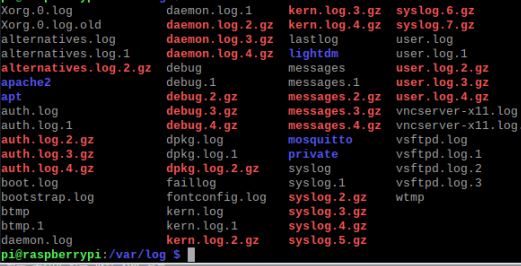
}

마지막에 usleep을 넣어주지 않아서 한 번 깜빡였다.



코드를 다 짜고 저장을 할려고 하니 장치에 남은 공간이 없다는 게 떴다.

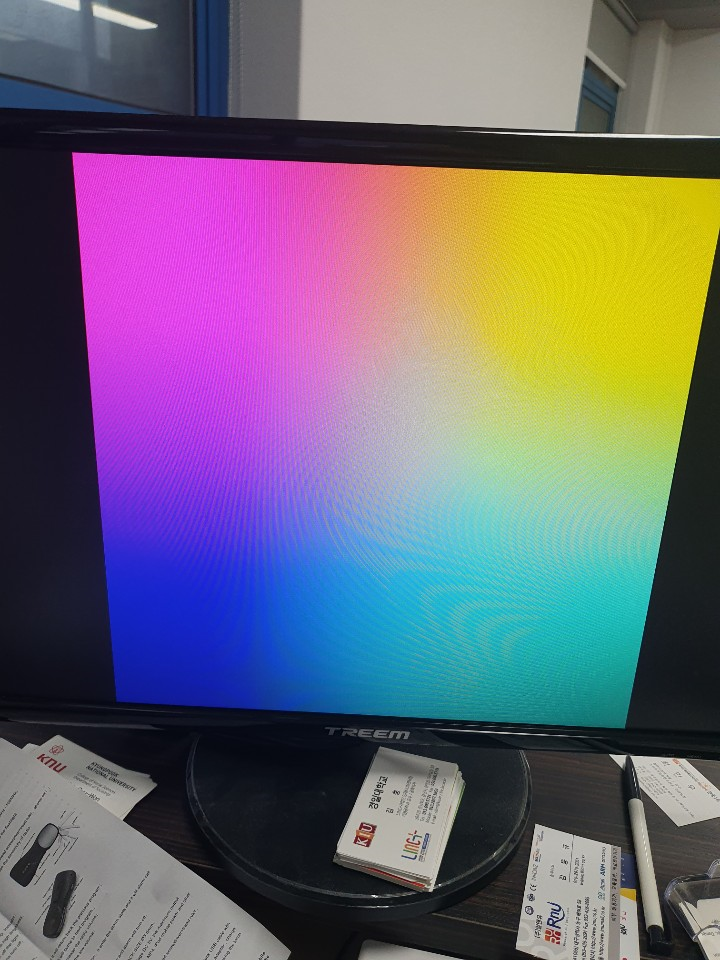
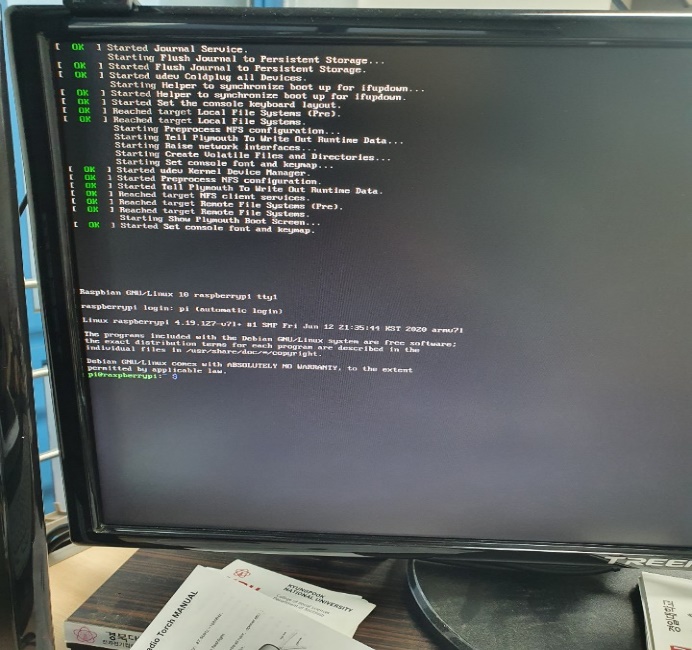


“df-h”를 사용하여 어느쪽에서 공간을 다 차지 하고 있는지 확인을 해본 결과 /dev/root가 공간을 다 차지하고 있다는 것을 알게 되었다.

로그들이 용량을 다 차지하고 있는 건가 싶어서 로그파일들을 다 삭제해봤지만 해결 되지 않았다.

그래서 구글에 검색을 해서 여러가지 방법이 있어서 적용을 해보니, vnc viewer에서 해상도가 1280\*720인데도 불구하고 cannot currently show라는 에러가 났다.

1. **고찰**

 코드를 짜고 실행을 하다 보니 저장공간이 부족하다는 경고문과 함께 main.c 저장이 되지 않았다. 그래서 인터넷에 검색하여 로그파일들을 삭제해보고 다른 파일들도 삭제하다가 vnc viewer 에러가 나서 모니터에 라즈베리파이를 연결해서 시작해보니 시작화면이 이렇게 뜨면서 진입이 되지 않았다. 계속 시도해보다가 도저히 뭐가 문젠지 찾아내지 못하고 sd카드를 포맷하고 다시 이미지 write를 했다. 필자가 생각하기로는 insmod로 커널 모듈을 적재하고 rmmod를 사용하여 커널 모듈을 제거해야 했는데 필자는 그 작업을 하지 않아서 저장 공간이 부족하다고 뜬 거라고 생각 한다. 이번 과제의 전체적인 흐름으로는 입출력 함수 처리를 위한 구조체를 선언해주고, read, write함수를 모듈에서 구현함으로써 유저 영역으로 데이터를 보내고 가져오고 하였다. 그리고 insmod를 사용하여 커널 함수를 호출하였고, dev\_t devno로 major와 minor를 전달하고 디바이스 파일의 번호를 받았다. map=ioremap(GPIO\_BASE, GPIO\_SIZE)를 사용하여 물리 주소 번지를 주면 가상 주소 번지를 알려줬고, gpio=(volatile unsigned int\*)map를 사용하여 gpio register base로 설정하였다. 그리고 end module같은 경우에는 문자 디바이스의 등록을 해제한 후 구조체를 해제하고 마지막으로 매핑된 메모리를 삭제해줬다. 그 과정에서 첫번째 과제는 유저모드에서 설정하는 것이니 main.c에서 write를 사용해서 led를 켜주면 됐고, 과제2번은 운영체제가 스위치의 상태 레지스터를 읽어서 명령의 수신 여부를 주기적으로 확인하였다. 그래서 main.c안에 while문을 사용하여 폴링을 하는 동안 다른 프로세스에게 CPU를 양도하지 않고 하드웨어 장치가 동작을 완료하는 동안 계속 루프를 돌면서 하드웨어 상태를 체크하게 한 후 led\_module.c에서 switch값에 따라 led가 불이 켜지고 꺼짐을 제어했다. 과제를 하면 할수록 느끼는 게 있다. 당연시 사용하던 마우스 클릭이나 키보드 입력과 같은 것들이 실제로 device driver가 있기에 우리가 컴퓨터에서 쉽게 사용할 수 있다는 것을 알게 되었다.

1. **프로그램 소스 파일**

**과제1**

**-main.c**

#include <stdio.h>

#include <string.h>

#include <fcntl.h>

#include <stdlib.h>

#include <unistd.h> //for sleep function usleep

void main(int argc, char\*\* argv){

char buf[BUFSIZ];

char i = 0;

char mode[BUFSIZ];

int fd = -1;

char led\_control[BUFSIZ];

fd =open("/dev/ledtest", O\_RDWR);

if(argc != 1){

printf("./main\n");

exit(0);

}

printf("Select the mode, A(blink), B: ");

scanf("%c", mode);

if (strcmp(mode , "A")==0)

{

for (int j=0; j<3; j++) {

write(fd, "1", 1); //turn on the led

usleep(500000); //wait 0.5 sec

write(fd, "0", 1); //turn off the led

usleep(500000); //wait 0.5 sec

}

}

if (strcmp(mode , "B") == 0) {

printf("Select 1(ON) or 0(OFF) : ");

scanf("%s", led\_control);

if (strcmp(led\_control,"1") == 0) write(fd, "1", 1); //turn on the led

else write(fd, "0", 1); //turn off the led

return 0;

}

printf("fd: %d\n", fd);

printf("mode type: %s\n", mode);

close(fd);

}

**-led\_module.c**

#include <linux/module.h>

#include <linux/cdev.h>

#include <linux/uaccess.h>

#include <linux/fs.h>

#include <linux/io.h>

#define GPIO\_BASE 0xfe200000

#define GPIO\_SIZE 256

#define GPIO\_SET(g) (\*(gpio+7)=1<<g)

#define GPIO\_CLR(g) (\*(gpio+10)=1<<g)

#define GPIO\_GET(g) (\*(gpio+13)&(1<<g))

#define GPIO\_IN(g) (\*(gpio+((g)/10)) &=~(7<<(((g)%10)\*3)))

#define GPIO\_OUT(g) (\*(gpio+((g)/10)) |=(1<<(((g)%10)\*3)))

#define GPIO\_MAJOR 243

#define GPIO\_MINOR 0

#define GPIO\_DEVICE "ledtest"

#define GPIO\_NUM 26

static int gpio\_open(struct inode\*, struct file\*);

static int gpio\_release(struct inode\*, struct file\*);

static ssize\_t gpio\_read(struct file\*, char\*, size\_t, loff\_t\*);

static ssize\_t gpio\_write(struct file\*, const char\*, size\_t, loff\_t\*);

volatile unsigned \*gpio;

static char msg[BLOCK\_SIZE] = {0};

struct cdev gpio\_cdev;

static struct file\_operations gpio\_fop ={

.owner = THIS\_MODULE,

.open=gpio\_open,

.release=gpio\_release,

.read=gpio\_read,

.write=gpio\_write,

};

int start\_module(void){ //insmod led\_module

unsigned int cnt=1;

static void\* map;

int add;

dev\_t devno;

printk(KERN\_INFO "START MODULE\n");

devno=MKDEV(GPIO\_MAJOR, GPIO\_MINOR);

register\_chrdev\_region(devno, 1, GPIO\_DEVICE);

cdev\_init(&gpio\_cdev, &gpio\_fop);

gpio\_cdev.owner = THIS\_MODULE;

add=cdev\_add(&gpio\_cdev, devno, cnt);

map=ioremap(GPIO\_BASE, GPIO\_SIZE);

gpio=(volatile unsigned int\*)map;

GPIO\_IN(GPIO\_NUM);

GPIO\_OUT(GPIO\_NUM);

return 0;

}

void end\_module(void){ //rmmod led\_module

dev\_t devno = MKDEV(GPIO\_MAJOR,GPIO\_MINOR);

unregister\_chrdev\_region(devno, 1);

cdev\_del(&gpio\_cdev);

if(gpio)

iounmap(gpio);

printk(KERN\_INFO "END MODULE\n");

}

static int gpio\_open(struct inode \*inode, struct file \*file){ //access a device driver

try\_module\_get(THIS\_MODULE);

printk("OPEN - gpio device\n");

return 0;

}

static int gpio\_release(struct inode\* inode, struct file\* file){ //int close system call, when you close the file

module\_put(THIS\_MODULE);

printk("CLOSE - gpio device\n");

return 0;

}

static ssize\_t gpio\_read(struct file\* file, char\* buf, size\_t len, loff\_t\* off){ //when you read the system call

int cnt;

cnt = copy\_to\_user(buf, msg, strlen(msg)+1);

printk("GPIO device READ: %s \n", msg);

return cnt;

}

static ssize\_t gpio\_write(struct file\* file, const char\* buf, size\_t len, loff\_t\* off){ //when you call system call and use

short cnt;

memset(msg, 0, BLOCK\_SIZE);

cnt = copy\_from\_user(msg, buf, len);

(!strcmp(msg, "0")) ? GPIO\_CLR(GPIO\_NUM):GPIO\_SET(GPIO\_NUM);

printk("gpio device WRITE: %s \n", msg);

return cnt;

}

MODULE\_LICENSE("GPL");

module\_init(start\_module);

module\_exit(end\_module);

**과제2**

**-main.c**

#include <stdio.h>

#include <string.h>

#include <fcntl.h>

#include <stdlib.h>

void main(int argc, char\*\* argv){

int fd = -1;

if(argc != 1){

printf("type ./main\n");

exit(0);

}

fd =open("/dev/ledtest", O\_RDWR);

while(1){ //infinite loading. wait until the getting switch value

write(fd, NULL, NULL);

}

close(fd);

}

**-led\_module.c**

#include <linux/module.h>

#include <linux/cdev.h>

#include <linux/uaccess.h>

#include <linux/fs.h>

#include <linux/io.h>

#define GPIO\_BASE 0xfe200000

#define GPIO\_SIZE 256

#define GPIO\_SET(g) (\*(gpio+7)=1<<g)

#define GPIO\_CLR(g) (\*(gpio+10)=1<<g)

#define GPIO\_GET(g) (\*(gpio+13)&(1<<g))

#define GPIO\_IN(g) (\*(gpio+((g)/10)) &=~(7<<(((g)%10)\*3)))

#define GPIO\_OUT(g) (\*(gpio+((g)/10)) |=(1<<(((g)%10)\*3)))

#define GPIO\_MAJOR 243

#define GPIO\_MINOR 0

#define GPIO\_DEVICE "ledtest"

#define GPIO\_NUM 26

static int gpio\_open(struct inode\*, struct file\*);

static int gpio\_release(struct inode\*, struct file\*);

static ssize\_t gpio\_read(struct file\*, char\*, size\_t, loff\_t\*);

static ssize\_t gpio\_write(struct file\*, const char\*, size\_t, loff\_t\*);

volatile unsigned \*gpio;

static char msg[BLOCK\_SIZE] = {0};

struct cdev gpio\_cdev;

static struct file\_operations gpio\_fop ={

.owner = THIS\_MODULE,

.open=gpio\_open,

.release=gpio\_release,

.read=gpio\_read,

.write=gpio\_write,

};

int start\_module(void){ //insmod led\_module

unsigned int cnt=1;

static void\* map;

int add;

dev\_t devno;

printk(KERN\_INFO "START MODULE\n");

devno=MKDEV(GPIO\_MAJOR, GPIO\_MINOR);

register\_chrdev\_region(devno, 1, GPIO\_DEVICE);

cdev\_init(&gpio\_cdev, &gpio\_fop);

gpio\_cdev.owner = THIS\_MODULE;

add=cdev\_add(&gpio\_cdev, devno, cnt);

map=ioremap(GPIO\_BASE, GPIO\_SIZE);

gpio=(volatile unsigned int\*)map;

GPIO\_IN(GPIO\_NUM);

GPIO\_OUT(GPIO\_NUM);

return 0;

}

void end\_module(void){ //rmmod led\_module

dev\_t devno = MKDEV(GPIO\_MAJOR,GPIO\_MINOR);

unregister\_chrdev\_region(devno, 1);

cdev\_del(&gpio\_cdev);

if(gpio)

iounmap(gpio);

printk(KERN\_INFO "END MODULE\n");

}

static int gpio\_open(struct inode \*inode, struct file \*file){ //access a device driver

try\_module\_get(THIS\_MODULE);

printk("OPEN - gpio device\n");

return 0;

}

static int gpio\_release(struct inode\* inode, struct file\* file){ //int close system call, when you close the file

module\_put(THIS\_MODULE);

printk("CLOSE - gpio device\n");

return 0;

}

static ssize\_t gpio\_read(struct file\* file, char\* buf, size\_t len, loff\_t\* off){ //when you read the system call

int cnt;

cnt = copy\_to\_user(buf, msg, strlen(msg)+1);

printk("GPIO device READ: %s \n", msg);

return cnt;

}

static ssize\_t gpio\_write(struct file\* file, const char\* buf, size\_t len, loff\_t\* off){ //when you call system call and use

short cnt;

memset(msg, 0, BLOCK\_SIZE);

cnt = copy\_from\_user(msg, buf, len);

(!strcmp(msg, "0")) ? GPIO\_CLR(GPIO\_NUM):GPIO\_SET(GPIO\_NUM);

printk("gpio device WRITE: %s \n", msg);

return cnt;

}

MODULE\_LICENSE("GPL");

module\_init(start\_module);

module\_exit(end\_module);

**Makefile**

obj-m = led\_module.o

KDIR:=/lib/modules/$(shell uname -r)/build

PWD:=$(shell pwd)

default:

$(MAKE) -C $(KDIR) SUBDIRS=$(PWD)

clean:

$(MAKE) -C $(KDIR) SUBDIRS=$(PWD)

1. **자료 출처**

<https://wikibook.co.kr/article/when-the-disk-is-full/>

<https://blog.naver.com/PostView.nhn?blogId=shinelife39&logNo=221422961749&parentCategoryNo=&categoryNo=17&viewDate=&isShowPopularPosts=true&from=search>

<https://dalgong2.tistory.com/6>

<https://jaebworld.tistory.com/27>